**Логический тип (bool) в Python**

Объект типа bool (от англ. *boolean — логическое значение*) принимает одно из двух значений — True (истина) или False (ложь). Обратите внимание: True и False записываются с большой буквы. Переменные типа bool используются для хранения значение типа да/нет, случилось/не случилось и т.п. Также операции сравнения возвращают результат типа boolean. Например, 3 > 5 возвращает False, а 3 < 5 — True.

**Логические операции**

not A — логическое "не" (отрицание). Истинно, если А ложно, и ложно, если А истинно.
A and B — логическое "и" (конъюнкция). Истинно тогда и только тогда, когда истинны A и B.
A or B — логическое "или" (дизъюнкция). Истинно, если хотя бы одно из А или B истинно.
A ^ B — исключающее "или". Истинно тогда и только тогда, когда истинен *ровно один* из аргументов.

**Условная инструкция в Python**

Рассмотрим программу с нелинейной структурой.

Допустим мы хотим по данному числу x определить его абсолютную величину (модуль). Программа должна напечатать значение переменной x, если *x*>=0 или же величину −*x* в противном случае. Линейная структура программы нарушается: в зависимости от справедливости условия *x*>=0 должна быть выведена одна или другая величина. Соответствующий фрагмент программы на Питоне имеет вид:

x = int(input())
if x >= 0:
    print(x)
else:
    print(-x)

В этой программе используется условная инструкция **if** (если). После слова **if** указывается проверяемое условие (*x*>=0), завершающееся двоеточием. После этого идет блок (последовательность) инструкций, который будет выполнен, если условие истинно; в нашем примере это вывод на экран величины x. Затем идет слово **else** (иначе), также завершающееся двоеточием, и блок инструкций, который будет выполнен, если проверяемое условие неверно, в данном случае будет выведено значение -x.

Итак, условная инструкция в Питоне имеет следующий *синтаксис*:

if Условие:
    Блок инструкций 1
else:
    Блок инструкций 2

Блок инструкций 1 будет выполнен, если Условие истинно. Если Условие ложно, будет выполнен Блок инструкций 2.

В условной инструкции может отсутствовать слово else и последующий блок. Например, если дано число x и мы хотим заменить его на абсолютную величину x, то это можно сделать следующим образом:

if x < 0:
    x = -x
print(x)

В этом примере переменной x будет присвоено значение -x, но только в том случае, когда x<0. А вот инструкция print(x) будет выполнена всегда, независимо от проверяемого условия.

Для выделения блока инструкций, относящихся к инструкции if или else в языке Питон используются *отступы*. Все инструкции, которые относятся к одному блоку, должны иметь равную величину отступа, то есть одинаковое число пробелов в начале строки. Рекомендуется использовать отступ в 4 пробела (и не рекомендуется использовать в качестве отступа символ табуляции).

Это одно из существенных отличий синтаксиса Питона от синтаксиса большинства языков, в которых блоки выделяются специальными словами, например, нц… кц в Кумире, begin… end в Паскале или фигурными скобками {…} в Си.

**Операторы сравнения**

Как правило, в качестве проверяемого условия используется результат вычисления одного из следующих операторов сравнения:

* <  Меньше — условие верно, если первый операнд меньше второго.
* >  Больше — условие верно, если первый операнд больше второго.
* <=  Меньше или равно.
* >=  Больше или равно.
* ==  Равенство. Условие верно, если два операнда равны.
* !=  Неравенство. Условие верно, если два операнда неравны.

Например, условие x \* x < 1000 означает “значение x \* x меньше 1000”, а условие 2 \* x != y означает “удвоенное значение переменной x не равно значению переменной y”.

Операторы сравнения в Питоне можно объединять в цепочки (в отличии от большинства других языков программирования, где для этого нужно использовать логические связки), например, a == b == c или 1 <= x <= 10.

Операторы сравнения возвращают значения специального логического типа **bool**.

**Принцип условного исполнения**

*Принцип условного исполнения*относится к базовым принципам, выдвинутым Джоном фон Нейманом в 1946 году.

Команды из программы не всегда выполняются одна за другой. Возможно присутствие в программе команд условного перехода, которые изменяют последовательность выполнения команд в зависимости от значений данных.

Впрочем, этот принцип был сформулирован задолго до фон Неймана Адой Лавлейс и Чарльзом Бэббиджем.



В приведенном примере Оператор 2 выполняется только в том случае, когда условие истинно.
На левой блок-схеме в случае когда условие ложно не выполняется ничего, а на блок-схеме справа — Оператор 3.
Оператор условного исполнения есть во всех императивных языках программирования.

**Стиль программирования (для Python)**

PEP 8 — документ, описывающий соглашение о том, как писать код на языке Python. PEP 8 создан на основе рекомендаций создателя языка Гвидо ван Россума.  Ключевая идея Гвидо такова: код читается намного больше раз, чем пишется. Собственно, рекомендации о стиле написания кода направлены на то, чтобы улучшить читаемость кода и сделать его согласованным между большим числом проектов. В идеале, если весь код будет написан в едином стиле, то любой сможет легко его прочесть.

**Основные правила PEP 8:**

**Форматирование**

* Используйте четыре пробела для отступов. Не делайте отступов в два пробела. Wing Ide помогает правильно расставлять пробелы. По умолчанию в Wing Ide клавиша Tab ставит четыре пробела.
* Пишите *import*  каждого модуля в отдельной строке
* Располагайте все *import*'ы в верхней части кода перед любыми глобальными объявлениями.
* Отделяйте блок *import*'ов от кода пустой строкой.
* Не используйте конструкцию *from … import \**}
* Скобки не отделяются пробелами с внутренней стороны. Между функцией и ее аргументами пробел не ставится.

*spam(ham[1], {eggs: 2})        # Правильно*
*spam( ham[ 1 ], { eggs: 2 } )  # Неверно*

* Перед запятой, двоеточием пробел не ставится, после -— ставится.

*if x == 4:*

*print(x, y)*

*x, y = y, x       # Правильно*

*if x == 4 :*

*print(x , y)*

*x , y = y , x     # Неверно*

* Всегда окружайте следующие бинарные операторы ровно одним символом пробела с каждой стороны:
	+ присваивания (=, +=, -= и т. д.),
	+ сравнения (==, <, >, !=, <>, <=, >=, in, not in, is, is not),
	+ логические (and, or, not),
	+ арифметические (+, -, \*, /, //, \%, \*\*).
* Не располагайте несколько инструкций в одной строке. Разнесите их по разным строкам.

*x = 3            # Правильно*
*func(10)*
*x = 3; func(10)  # Неверно* Не располагайте блок из нескольких инструкций на той же строке сразу после двоеточия (после *if*, *while* и т. д.)

**Комментарии**

* Комментарии, противоречащие коду, хуже, чем их отсутствие.
* Располагайте однострочные комментарии после кода в той же строке и отделяйте их от кода не менее чем двумя пробелами. Комментарии должны начинаться с # и одного пробела.

**Имена**

* Не используйте символы `l', `O', и `I' как имена переменных. В некоторых шрифтах они могут быть очень похожи на цифры.
* Имена переменных и функций должны содержать только маленькие буквы. Слова разделяются символами подчёркивания. Примеры:

*name, name\_with\_several\_words\_in\_it*

* Имена констант должны содержать только заглавные буквы. Слова разделяются символами подчёркивания. Примеры:

*NAME, NAME\_WITH\_SEVERAL\_WORDS\_IN\_IT*

* Давайте переменным говорящие английские имена, не используйте транслит.

*num\_letters = int(input())         # Правильно*

*kolvo\_bukv = int(input())          # Неверно*

**Функции**

* Разделяйте определения функций двумя пустыми строками. Отделяйте определения функций от основного кода двумя пустыми строками до и после.
* Разделяйте тело функции на логические части одной пустой строкой.